jamk.fi

DIY-Platformer

Kuvasta kentäksi

Topias Paasonen

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Paasonen, Topias	Opinnäytetyö, AMK	toukokuu 2017
	Sivumäärä 22	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
DIY-Platformer
Kuvasta kentäksi

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Rantala, Ari

Toimeksiantaja(t)

Tiivistelmä

Projektissa kehitettiin tasoloikkapeli, jossa pelimekaniikat ovat yksinkertaiset. Pohjatutkimukset keskittyivät selvittämään, miten paperipiirroksia ja piirto-ohjelmilla piirrettyjä kuvia voidaan hyödyntää kenttien luomisessa. Tavoitteena oli julkaista peli, joka sisältää yksinkertaisen kenttäeditorin, jolla kuka tahansa voi luoda ja muokata omia kenttiä piirtämällä kentän paperille ja ottamalla paperista kuvan mobiililaitteellaan.

Työn toteutus tehtiin useaan otteeseen. Ensin luotiin versio Windows-puhelimille käyttämättä valmista pelimoottoria, sitten Android-laitteille suunnattu versio käyttäen LibGDX-pelimoottoria ja Android-kehitykseen suositeltua Android Studiota. Lopullinen versio toteutettiin Unity Enginellä sovelluskehittimenä Visual Studio 2017.

Kenttäeditorin mahdollisuuksia tutkittaessa toteutus päätettiin tehdä käyttäen AForge.NET-kirjastoa. Kenttäeditori lukee annetusta kuvasta, pikselitietoa käyttäen, piirretyt elementit ja muodostaa niistä pelimoottorille ymmärrettäviä objekteja. Kentän hienosäätö, kuten aloituspisteen ja maalin sijoittaminen tapahtuu ohjelman käyttöliittymän kautta.

Opinnäytetyön tuloksena oli pelin prototyyppi, joka osoittaa kenttien luonnin kuvista olevan mahdollista, muttei ole kokonainen, valmis tuote. Pelistä puuttui monia ominaisuuksia, jotta sitä olisi voinut kutsua valmiiksi. Muun muassa suunnitelmaan kuuluneita ominaisuuksia kuten pelaajahahmon muokkausta ei toteutuksessa ollut ollenkaan.

Tulosta ei jatkettu eteenpäin näyttäväksi prototyypiksi, vaan aikaiseksi saatuun tulokseen tyydyttiin. Mutta mikäli työtä kehitetty valmiiksi tuotteeksi asti, se olisi aloitettu uudestaan käyttäen modernia kuvankäsittelykirjastoa.

Avainsanat (asiasanat)
Pelinkehitys, tasoloikka, kenttäeditori, kuvankäsittely, DIY-Platformer

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Paasonen, Topias	Bachelor's thesis	May 2017
		Language of publication: Finnish
	Number of pages: 22	Permission for web publication: x

Title of publication

DIY-Platformer

From a picture to a level

Degree programme

Software Engineering

Supervisor(s)

Rantala, Ari

Assigned by

Abstract

The thesis' subject was developing a platformer game with simple game mechanics and a robust level editor. Research focused on figuring out how paper and computer-drawn drawings could be used in level creation. The goal was to publish a full game including a simple level editor, which is easy to use for anyone: draw a level on a paper and take a picture of it using your mobile device.

The implementation was made multiple times using different platforms. First version was made for Windows-phones without using a proper game engine. The next version was for Android-devices using LibGDX as a game engine and the official Android development environment Android Studio. The final version was developed using Unity Engine and Visual Studio 2017.

Research showed that the level editor should use AForge.NET-framework. The level editor reads pixel data from the given picture and using the shapes drawn to create objects for the game engine. Rest of the level, including the starting point and goal, are placed using the user interface of the program.

The result of the thesis is a game prototype, that shows that creating levels from pictures is possible and feasible, but it is not a published product. The game was not developed further afterwards. If the game would have been developed into a full product, the development would have been started from the beginning using a modern image processing tools.

Keywords/tags (subjects)

Game development, platformer, level editor, image processing, DIY-Platformer

Miscellaneous

Sisältö

Kuviot

Käsitteet

Adom

1994 julkaistu ilmaispeli, joka käyttää ASCII-grafiikkaa pelimaailman ja käyttöliittymän kuvaamiseen.

Android

Googlen Linux-pohjainen käyttöjärjestelmä älypuhelimille ja muille mobiililaitteille.

Android Studio

Androidin virallinen ohjelmointiympäristö.

ASCII-grafiikka

Graafinen ratkaisu, joka koostuu kirjaimista, numeroista ja erikoismerkeistä.

Blobi

AForgen Imaging-kirjaston luokka, joka edustaa kuvan osaa.

BT.709

International Telecommunication Unionin radiokommunikaatio sektorin standardi korkearesoluutioiselle televisiolle, mukaan lukien Luma kertoimet.

Catalano-framework

AForgea ja Accord.NET:a Javalle ja Androidille kääntävä kirjasto.

Cocos2D

C++ pohjainen, avoimen lähdekoodin monialustainen ohjelmistokehys.

ConnectedComponentsLabeling-suodatin

AForgen suodatin, joka jaottelee kuvan objektit blobeiksi.

Collideri

Pelimoottorin törmäystarkastelun osa, joka voi törmätä muihin collidereihin.

FlatAnglesOptimizer-luokka

AForgen luokka, joka yhdistää muodon lähes yhdensuuntaiset reunat yhdeksi suoraksi reunaksi.

LGPL v3

Lisenssi, joka sallii kaupallisen käytön.

Graham-algoritmi

Pistejoukon kuperan kuoren etsivä algoritmi.

Harmaa sävytys

Kuvan muokkaus, jonka lopputuloksena kuvan pikseleillä ei ole väriä, vaan ainoastaan intensiteetti.

iOS

Applen käyttöjärjestelmä, joka on käytössä Applen mobiililaitteissa ja Apple TV:ssä.

IntPoint

AForgen käyttämä struktuuri, jossa on kaksi kokonaisluku koordinaattia.

Invert-suodatin

AForgen suodatin, joka kääntää kuvan värit päinvastaisiksi alkuperäiseen kuvaan verrattuna.

JBox2d

C++ pohjaisen Box2D fysiikka moottorin Java käännös.

Keko koko

Androidin ohjelmalle käyttöön antama muistin määrä, joka riippuu laitteen käytössä olevan muistin määrästä.

Kenttäeditori

Ohjelma tai osa ohjelmaa, jolla voi luoda uusia kenttiä.

Käyttäjärajapinta

Ohjelman osa, joka on vastuussa käyttäjälle annettavasta ja käyttäjän antamasta tiedosta.

LineStraightenингOptimizer-luokka

AForgen muodon kulmia optimoiva luokka, joka poistaa pisteitä, jotka ovat lähellä kulmaa tehden kulmista suorempia.

Pelimekaniikka

Pelin sääntö tai ominaisuus. Pelimekaniikat määrittelevät pelin.

Pelimoottori

Ohjelmistokehys peleille, joka on vastuussa useista peleille olennaisista komponenteista kuten fysiikkamallinnuksesta.

Polarisointi

Kuvanmuokkaus kaksi väriseksi yleensä mustaksi ja valkoiseksi.

Polygoni collideri

Unityn polygonimuotoinen collideri.

RAM

Nopea työmuisti.

Reuna collideri

Unityn collideri, joka muodostaa yksittäisen reunan.

RGB

Värimalli, jossa värit muodostuvat punaisesta, vihreästä ja sinisestä.

System.Drawing.Bitmap

.NET ohjelmistokehyksen luokka, joka hallinnoi kuvan pikselidataa.

Tasoloikkapeli

Peligenre, jossa pelaajahahmo liikkuu kentällä olevia platformeja eli tasoja pitkin.

Threshold-suodatin

AForgen suodatin, joka polarisoi harmaa sävytetyn kuvan.

Unity3D

Moderni, monialustainen pelimoottori, jota voidaan käyttää kaksi- ja kolmiulotteisten pelien kehityksessä.

Vector2

Unityn luokka, joka sisältää kaksiulotteisen vektorin.

1 Johdanto

Yhden hengen peliprojektit ovat aina erittäin haastavia, varsinkin jos pelinkehittäjällä on jokin selvästi heikko osaamisala. Tyypillisesti näissä tapauksissa on joko tyydyttävä heikkoon laatuun kyseisessä osassa lopputuotetta, hankittava apua joko toiselta kehittäjältä tai käyttäen muiden valmiita tuotteita tai kehitettävä peli, jossa kehittäjän heikkous ei näy lopputuloksessa.

Ennen työn aloittamista heikkoudeksi todettiin graafinen osaaminen. Graafisen osaamattomuuden peittäminen peliprojektissa on haastavaa ovathan videopelit visuaalista mediaa. Peli voitaisiin tehdä hyödyntäen ASCII-grafiikkaa Adom-tyyliin, mutta kyseinen tyyli on pieniä harrastepelejä pullollaan. Modernina vaihtoehtona on hyödyntää muiden tekemää valmisgrafiikkaa, mutta se ole houkutteleva tapa. Projektissa päädyttiin hyödyntämään erilaista modernia tekniikkaa: kuvien ja muotojen tunnistusta.

Työssä tehtiin tasoloikkapeli, jonka kentät luodaan ottamalla kuva paperille piirretystä kentästä. Idea syntyi kaupunkimaisemaa katsellessa: Miksi olla käyttämättä kerrostaloista, teistä ja puistoista luonnostaan koostuvia kenttiä? Tutkimuksen alkuvaiheessa todettiin kokonaisen maisemakuvan käytön olevan epäkäytännöllistä. Kenttien luonti rajattiin käyttämään paperille piirretyistä kentistä otettuja valokuvia sekä piirto-ohjelmalla piirrettyjä kenttiä.

2 Suunnitelma

DIY-Platformer on pohjiltaan yksinkertainen tasoloikkapeli pääasiallisena alustana mobiililaitteet. Pelimekaniikat pyritään pitämään kaikille pelaajille tuttuina, mutta lopullisen tuloksen mahdollisuutta kehittyä monimutkaisemmaksi ei tahdota rajoittaa. Pelin vetonaulana toimii joustava ja tehokas kenttäeditori, jolla kuka tahansa voi luoda täysin kustomoituja kenttiä ja jakaa niitä ystävilleen pelattavaksi.

Peli itsessään ei kerro minkäänlaista tarinaa, eikä siinä ole valmista kattavaa kampanjaa. Peli suunniteltiin sisältävän vain muutaman valmiin tason, jotka valikoitaisiin parhaiden beta-version tasojen joukosta. Pelin sisältö jäisi pelaajien yhdessä yhteisönä luotavaksi.

2.1 Tasoloikka

Pelin tavoitteena on kulkea kentällä olevien esteiden kuten vihollisten ja rotkojen ohi maaliin, jolloin pelaajan suoritus pisteytetään. Yksinkertainen kentän läpäisyaikaan pohjautuva pisteytys mahdollistaa tulosten vertailun pelaajien kesken. Peli koostuu pelaajahahmosta, vihollisista sekä platformeista, joiden päällä pelaaja voi liikkua.

DIY-Platformerin tasoloikkamekaniikat ovat minimalistiset:

- Liikkuminen vasemmalta oikealle
- Hyppääminen ylöspäin
- Viholliseen törmääminen aiheuttaa kentän alkamisen alusta
- Kentältä ulos tippuminen aiheuttaa kentän alkamisen alusta
- Maalin saavutus antaa kuluneen ajan mukaisen pistemäärän ja läpäisee kentän

Pelimekaniikoita voidaan tarvittaessa laajentaa sisältämään seuraavia elementtejä:

- Liikkuvat ja ampuvat viholliset
- Vihollisten ampuminen ja tuhoaminen
- Liikkuvat platformit
- Välimaalit, joissa pelaajan on käytävä ennen varsinaisen maalin saavuttamista
- Kyltit, joihin kulkemalla pelaajalle näytetään teksti muotoinen viesti, jossa voidaan antaa vinkkejä kentän läpäisyyn tai kertoa kenttää koskevaa tarinaa
- Kenttään sijoitettavat voimalisät, jotka antavat pelaajalle väliaikaisia voimia kuten tuplahypyn, suuremman hyppyvoiman tai vahvemmat ammukset
- Monimutkaisempi pisteytys, joka ottaa huomioon tuhotut viholliset, poimitut voimalisät, saavutetut välimaalit sekä kyltit

2.2 Kenttäeditori

Kenttäeditoria voi käyttää kuka tahansa ilman monimutkaisten ohjeiden opiskelua. Piirrä kenttä paperille, ota siitä valokuva, lisää maali-, sekä aloituspiste ja olet valmis pelaamaan omaa kenttääsi. Keskivertokäyttäjä valitsee pelaajahahmon valmiista vaihtoehdoista, säätää hahmon liikkumisnopeutta ja lisää kentälleen vihollisia vai-

keuttamaan pelaajan etenemistä. Kehittyneet käyttäjät ottavat valikot täyteen käyttöönsä: Luovat oman pelaajahahmonsa ja viholliset käyttäen samaa tekniikkaa kuin kentän luomisessa. Säätävät pelaajan hyppyvoimaa ja massaa. Lisäävät kentälleen taustakuvan, jossa platformit sulautuvat hienosti ympäristöönsä. Skaalaavat kenttänsä sopivalle skaalalle, jotta näytöllä näkyy vain halutut asiat kerrallaan.

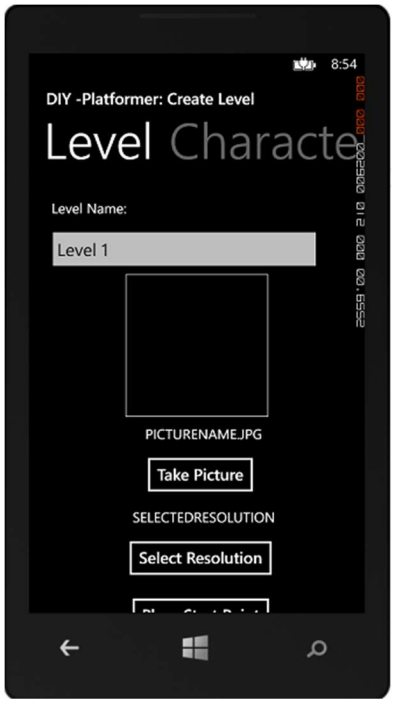
Kenttäeditorin tärkein ominaisuus on helppokäyttöisyys. Jos osaat käyttää kameraa ja ymmärrät kentän vähimmäisvaatimukset, saat tehtyä persoonallisen kentän.

3 Kehitysprosessi

3.1 Ensimmäinen versio

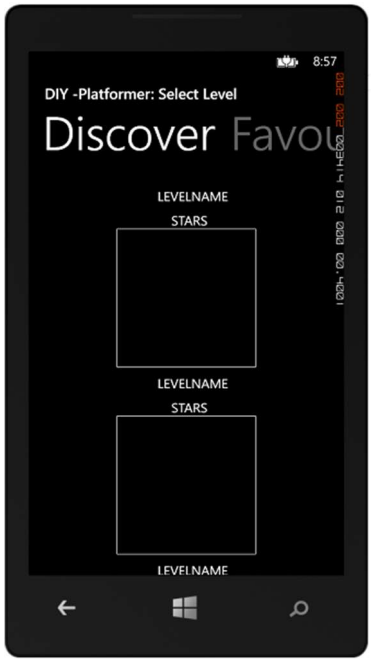
Ensimmäinen, niin sanottu “proof of concept”-versio tehtiin Windows Phone Programming -kurssin harjoitustyönä käyttäen C#-kieltä Visual Studio 2012:lla. Käyttäjärajapinta toteutettiin XAML:lla. Pohjatutkimuksen perusteella kuvien manipulointiin valittiin AForge.NET kirjasto, jolla kentästä otettu kuva polarisoitiin ja saadusta musta-valkokuvasta etsittiin pistekimput, jotka tulkittiin tasolle fyysisiksi objekteiksi eli platformeiksi, joiden päällä pelaajahahmo pystyy liikkumaan.

Käyttöliittymä tehtiin mustaa taustaa ja valkoista tekstiä ja nappuloita käyttäen (ks. kuvio 1). Huomattavaa on myös, että valikot suunniteltiin laite pystysuunnassa käytettäviksi, mutta itse pelin pelaaminen vaakatasossa. Kentän luonti -valikossa liikuttiin Taso – Hahmo – Vihollinen – Taustakuva – Yleisnäkymä kategorioiden välillä pyyhkäisemällä sivusuunnissa. Pelatessa hahmoa hallittiin ruudulla olleiden kahden pienen nuolinäppäimen ja A- sekä B-nappuloiden avulla.



Kuvio 1. Ensimmäisen version menu

Peliin suunniteltiin myös järjestelmä kenttien jakamiseksi ystävien ja tuntemattomien kesken Internetin välityksellä (ks. kuvio 2). Järjestelmään kuului tasojen arvostelu viiden tähden järjestelmää käyttäen sekä suodatusvalinnat hyviä arvosanoja saaneille kentille.



Kuvio 2. Suunniteltu Discover-jono muiden tekemien kenttien löytämiseksi

Suurimmaksi ongelmaksi toteutuksessa ilmeni törmäysten tarkistelu. Kuvista tulkitut pistekimput olivat pikselitarkkoja ja kuvat suuritarkkuuksisia, jolloin pikselitarkka törmäyksien tarkkailu ei sovi toteutukseen hitautensa vuoksi, varsinkaan mobiililaitteilla pelatessa. Ongelman ratkaisuun käytettiin jokaista pistekimppua ympäröivää nelikulmiota pikselitarkan tarkkailun rajoittamiseen ainoastaan niille pistekimpuille, joiden ympäröivien neliöiden sisällä pelaajahahmo on, mutta tämäkin ratkaisu oli liian raskas mobiililaitteille. Lopullinen ratkaisu oli hyödyntää ainoastaan ympäröiviä nelikulmioita, jolloin kaikkien platformien täytyi kentissä olla nelikulmioita. Toteutukseen ei oltu tyytyväisiä, mutta harjoitustyön koon huomioon ottaen siihen tyydyttiin.

Ensimmäisestä toteutuksesta todettiin, että lopullinen toteutus tulee tarvitsemaan kunnollisen pelimoottorin, joka huolehtii törmäyksien tarkkailusta, jotta peli saadaan tarpeeksi kevyeksi kohde laitteille. Kurssin vaatimasta Windows Phone 8-alustasta päätettiin luopua. Uudeksi alustaksi valittiin suositummat Android-laitteet.

3.2 Android versio käyttäen LibGDX-kirjastoa Android Studiolla

Seuraavassa vaiheessa peliä aloitettiin kehittää opinnäytetyönä. Uutena alustana Android, ohjelmointikieleksi vaihtui Java ja työvälineeksi valittiin Android Studio. Pelimoottoriksi löytyi useita vaihtoehtoja, mm. JBox2d, Cocos2D sekä Unity3D. Vaihtoehdoista parhaaksi todettiin LibGDX sen vahvan törmäyksien tarkistelun ja hyvän iOS-käännettävyyden ja aktiivisen yhteisön vuoksi. Muita vaihtoehtoja karsi esimerkiksi Cocos2D:n heikko dokumentaatio sekä iOS-keskittyneisyys ja Unity3D:n kolmiulotteisuus ja kooltaan yliampuvuus.

LibGDX ja Android Studio toimivat odotetusti, mutta ongelmina ilmeni Android-ohjelmien rajallinen RAM-muistin määrä. Suuriresoluutioisen kuvan datan käsittely RAM-muistissa vaati liian suuren määrän muistia, jopa maksimi kekokoolla (android.largeHeap=”true”). Käyttöliittymän luonti oli kömpelöä, mutta ei erityisen haastavaa. AForge.Net-kirjastoa ei voitu käyttää suoraan Android-versiossa, mutta siitä löytyi suurimmaksi osaksi Javalle käännetty versio Catalano-framework, joka valittiin kuvan käsittelykirjastoksi riittävän laajan Java-pohjaisen kirjaston puutteen vuoksi.

Suurin ongelma ilmeni kuvasta tulkitun datan siirtämisessä pelimoottorille käytettäväksi. Kuvan ja pelimoottorin koordinaatistojen erot aiheuttivat ongelmia, eikä kuvasta tulkittuja collidereita saatu täsmäämään taustalle asetettuun kuvaan (ks. kuvio 3). Kesällä 2015 kehittäjän huonosta taloudellisesta tilanteesta johtuen DIY-Platformerin kehitys keskeytyi, kunnes vuoden 2016 lopussa projekti aloitettiin uudelleen.

Kuvio 3. Vasemmalla lähdekuva, jossa pallo pelaajahahmona. Oikealla platformit pelimoottorin näkeminä, pystysuora palkki on kuvan alareuna.

3.3 Siirtyminen Unityyn

Aikaisemman LibGDX:n sijaan paremmaksi pelimoottoriksi todettiin Unity3D. Vaikka Unityä pidettiin aiemmin yliampuvana ja kolmiulotteisiin peleihin keskittyneenä, sen helppokäyttöisyys ja varsinkin yksinkertainen käyttöliittymän luonti, sekä kattava dokumentaatio osoittautuivat erittäin hyödyllisiksi ominaisuuksiksi. Suurin syy siirtymiselle Unityyn oli kuitenkin kehittäjälle kertynyt kokemus Unityn käytöstä työn ollessa tauolla.

LibGDX:llä työskennellessä eri alustoiden tukemiseksi jouduttiin tekemään töitä huomattavissa määrin, vaikka kaikille halutuille alustoille tuki olikin. Esimerkiksi, kameran käyttö erialustoilla oli huomattavan haastavaa. Unityllä alustojen väliset erot ovat tarvittaessa helppo eritellä käyttämällä yksinkertaisia tunnisteita. Unityä käyttäessä päästiin käyttämään myös C#:a ja siten Catalano-frameworkin sijaan alkuperäistä AForgeNet:a välttäen mahdolliset ongelmat käännöksen kanssa. Unityn laajuus helpottaisi jatkokehitystä, esimerkiksi helpolla mainostuella, joka oli suunniteltu käytettäväksi liiketoimintamalliksi. Käytännössä siirtyminen onnistui hyvin, tärkein logiikka

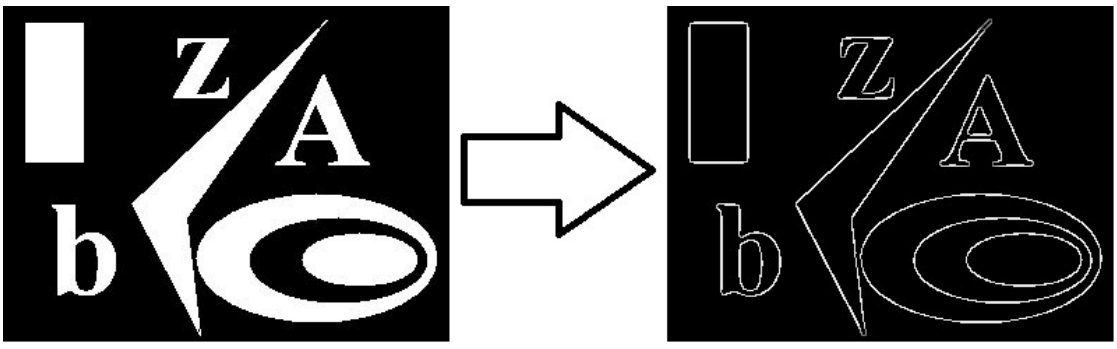
kentän luonnista oli helposti siirrettävissä ja Unityn pelimoottori hoiti pelin yksinkertaiset mekaniikat.

4 Kuvankäsittelykirjastot

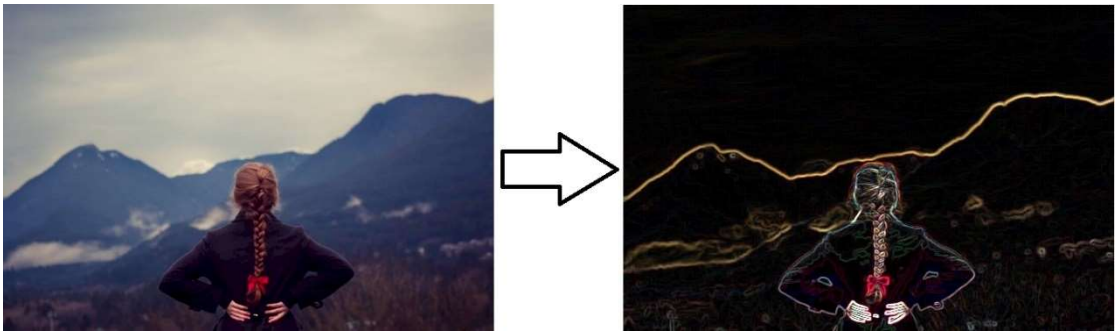
Kuvankäsittely on raskasta, eivätkä natiivit luokat kykene projektin vaatimaan manipulointiin. Tästä johtuen joudutaan käyttämään kuvankäsittelykirjastoa. Projektia aloittaessa useimmat riittävän kehittyneet kuvankäsittelykirjastot olivat maksullisia tai liian tiukasti lisensoituja. Käyttöön otettavan kirjaston tulisi olla ilmainen ja lisensoitu sallimaan kaupallisissa ohjelmissa käytön.

Ainoa riittävän vapaa vaihtoehto oli AForge.NET. AForge.NET on avoimen lähdekoodin kirjasto, joka on julkaistu LGPL v3 lisenssillä. AForgeen sisältyy kuvankäsittelyn lisäksi mm. koneoppimista ja robotiikkaa koskevat kirjastot. Nykypäivänä AForgea tuskin valittaisiin, sillä se on vanhentunut, viime päivityksestä on jo vuosia ja parempia vaihtoehtoja on tullut markkinoille. (Kirillov, 2013.)

Yksi näistä paremmista vaihtoehdoista on ImageProcessor. AForgen pystyessä yksin kertaiseen reunojen tunnistukseen ja yleiseen kuvan muokkaamiseen (ks. kuvio 4), ImageProcessor pystyy jopa mahdollistamaan maisemakuvista kenttien luomisen (ks. kuvio 5). (South.)



Kuvio 4. AForgen Canny Edge Detection



Kuvio 5. ImageProcessorin Prewitt Edge Detection

5 Kuvasta kentäksi

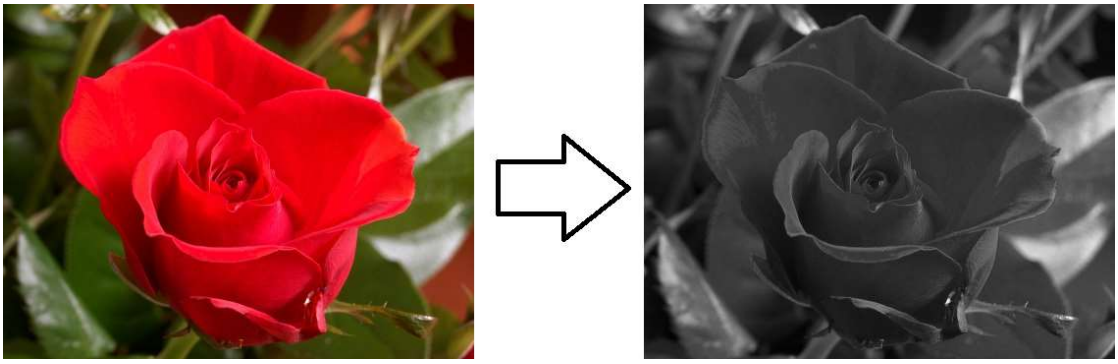
Suurin innovaatio DIY-Platformerissa on kenttien luonti kuvista. Kuvat voivat olla joko valokuvia paperille piirretystä kentästä tai piirto-ohjelmalla tehtyjä kenttiä. Kuvat koostuvat eri värisistä pikseleistä, joista värit tarkoittavat joko tyhjää (valkoinen) tai osaa tasosta (muut värit). Tällä jaolla saadaan kuvan koordinaatistossa joukko pisteitä, joiden kuuluu olla fyysisiä paloja tasossa.

5.1 Kuvan muokkaus, polarisaatio

Ensimmäinen askel on kuvan datan lataaminen sopivassa muodossa, tässä tapauksessa AForgelle toimivassa System.Drawing.Bitmap-muodossa. Mobiiliversiossa käyttäjä ottaa valokuvan laitteensa kameralla ja hyväksyy kuvan. Ohjelma tarvittaessa muokkaa kuvan tarkkuutta, jotta kuva ei ole liian suuri muistissa, ja ottaa sen käsiteltäväksi tavuina. Windows versiossa käyttäjä piirtää kuvan haluamallaan piirto-ohjelmalla tai luo kuvan jollain muulla tavalla, kuten ottamalla kuvan mobiiliversion tapaan. Kuva valitaan painamalla Take Picture -nappulaa ja antamalla kuvan sijainti.

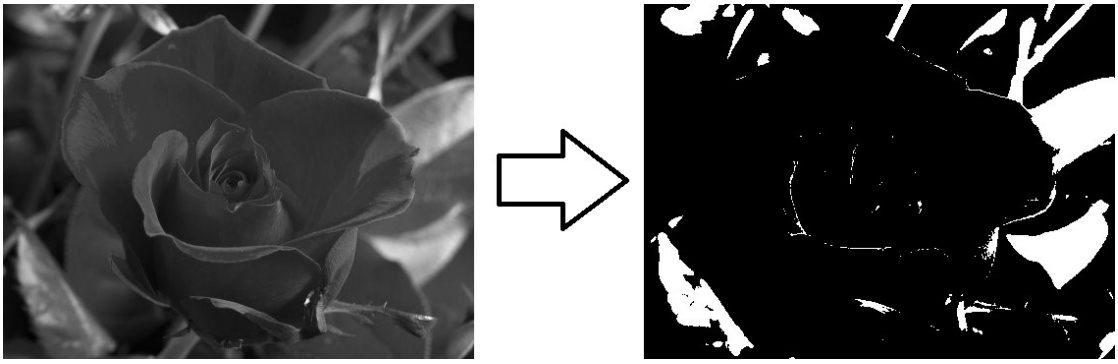
Seuraavaksi kuvan data muokataan edellä mainittuihin joukkoihin, tason osiin ja tyhjään. Kuvasta poistetaan värit, eli se harmaa sävytetään, jonka seurauksena kaikki pisteet ovat värittömiä harmaan sävyjä (ks. kuvio 6). RGB-tyyppisen värikuvan muuttaminen mustavalkoiseksi onnistuu yksinkertaisimmillaan ottamalla keskiarvo värien arvoista ja antamalla tulos kaikkien kolmen värin arvoksi. Tämä sävytys ei kuitenkaan ota huomioon eri värien luminanssia, ihmisten kykyä nähdä eri värejä paremmin kuin toisia. Jotta harmaan sävyt ovat ihmiselle oikean näköiset ja siten antavat paremman

tuloksen polarisaatioon, annetaan väreille omat painoarvot. Tässä tapauksessa käytettiin BT.709 -algoritmin mukaisia arvoja (Punainen = 0,2126; Vihreä = 0,7152 ja Sininen = 0,0722) (International Telecommunication Union 2015, 4).



Kuvio 6. Värikuvasta harmaa sävytetyksi

Näistä harmaan sävyistä ainoastaan riittävän tummat kohdat hyväksytään fyysisiksi osiksi. Riittävän tumman raja määritetään suodattamalla kuva Threshold-suodattimella. Suodattimelle annetaan intensiteetin raja-arvo, jota suuremmat arvot hyväksytään ja muutetaan mustiksi, pienemmät hylätään ja muutetaan valkoisiksi (ks. kuvio 7). Kahdeksan bittisille kuville raja-arvo on nollan ja 255 välillä, useimmissa tapauksissa sopiva arvo on 65 ja sadan välillä riippuen kuvan valoisuudesta ja käytetyn piirtovälineen vahvuudesta. Oletusarvoksi valittiin seitsemänkymmentä ja arvo suunniteltiin säädettäväksi kehittyneemmille käyttäjille.



Kuvio 7. Harmaa sävytetystä polarisoiduksi

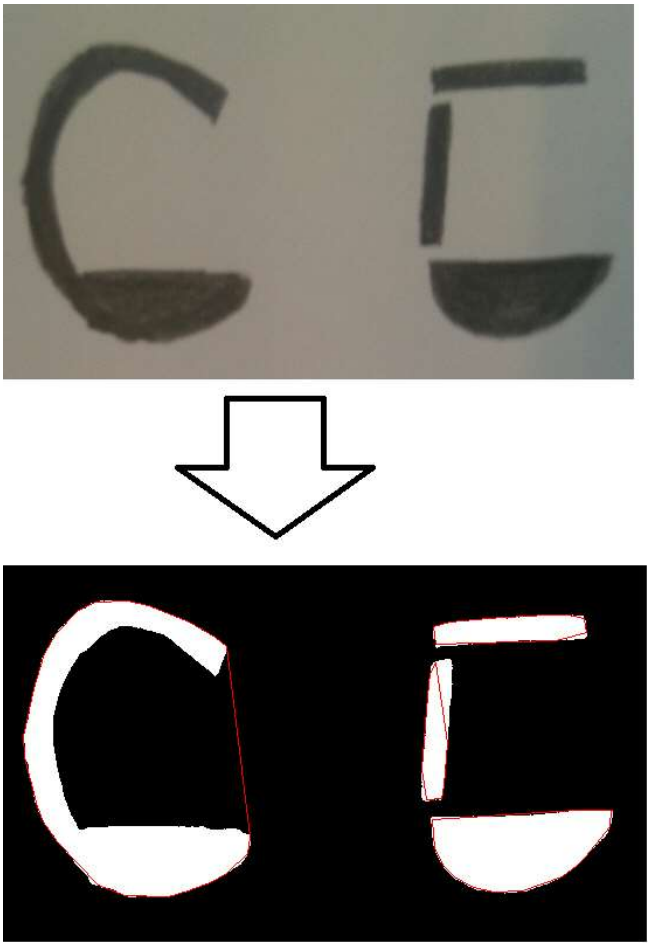
Tuloksena on kuva, jossa on mustaa ja valkoista, eli kuva on polarisoitu. Tästä polarisoidusta kuvasta voisi ottaa jokaisen mustan pisteen ja tehdä niistä tason, mutta

tämä olisi erittäin raskasta. Jokainen pikseli olisi yksittäinen objekti, joten kuvan dataa täytyy vielä muokata.

5.2 Pisteiden jaottelu

Käytännössä taso koostuu piirretyistä, eri muotoisista kuvioista eli platformeista. Polarisoidussa kuvassa platformit ovat toisiaan koskettavia pisteitä eli pistekimppuja. AForgessa nämä pistekimput tunnetaan Blobleina, jotka saadaan eriteltyä kuvasta käyttäen ConnectedComponentsLabeling-suodatinta. Blobien jaottelussa taustavärin täytyy olla musta, kaikki muun väriset pisteet lasketaan osaksi Blobeja, joten kuvan värit täytyy kääntää ympäri Invert-suodattimella. Näistä Blobeista ensin poistetaan erittäin pienet, virheellisesti luodut kappaleet. Pieniä, vahingollisia Blobeja syntyy, kun paperille piirretystä tasosta otetun valokuvan valotus ei ole täydellinen ja kuvaan syntyy tummempia kohtia.

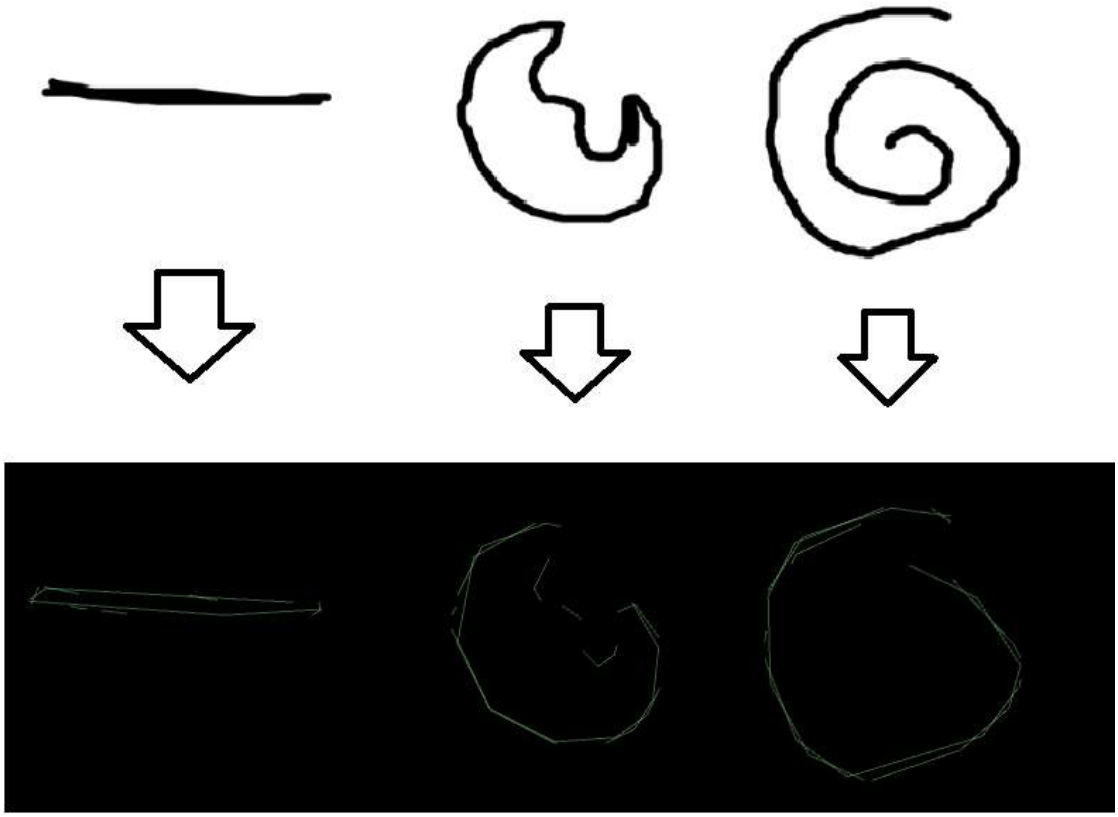
Blobeista voidaan tehdä pelimoottorille ymmärrettäviä objekteja eri tavoin. Niihin voidaan käyttää esimerkiksi Grahamin algoritmia, jolla saadaan pistejoukon kovera runko, mutta silloin kuperista kaarista tulee suoria viivoja (ks. kuvio 8). Yksinkertaisin vaihtoehto olisi järjestää reunapisteet järjestykseen ja tehdä niistä reuna- tai polygoni collidereita, mutta silloin colliderin sivujen määrä olisi järkyttävän suuri, jokainen piste kuuluisi kahteen sivuvektoriin.



Kuvio 8. Kupera ja kovera muoto ja niille koverat rungot

DIY-Platformerille sopivimmaksi tavaksi todettiin seuraava tapa: Otetaan Blobin ylä- ja alareunat sekä vasemmat ja oikeat reunat, käyttäen BlobCounter.GetBlobsTopAndBottomEdges() ja BlobCounter.GetBlobsLeftAndRightEdges()-metodeja. Nämä metodit antavat molempien akselien ensimmäiset pisteet molemmista suunnista järjestyksessä vasemmalta oikealle ylä- ja alareunoille ja ylhäältä alas vasemmalle ja oikealle reunalle. Seuraavaksi jokaisen reunan pisteistä muodostetaan EdgeCollidereita, ja jos seuraava piste on kaukana edellisestä pisteestä, aloitetaan uusi collideri. Ennen kuin pisteet tallennetaan collidereina, AForgen IntPoint-muuttujat täytyy muuttaa Unityn ymmärtämiksi Vector2:ksi ja pisteet käännetään Unityn koordinaatistoon ja skaalataan pelimoottorille. Samalla optimoidaan pisteet suoriksi LineStraightenOptimizer- ja FlatAnglesOptimizer-luokilla. Tällä tavalla heikkoudeksi jäävät ti-

lanteet, joissa platformin ulkoreunoihin kuuluu osia, jotka eivät ole Blobin ulommaisia osia (ks. kuvio 9). Tämän ongelman ratkaisemiseksi tällaiset Blobit voitaisiin pilkkoa osiin, mutta tämä päätettiin jättää jatkokehitykseen.

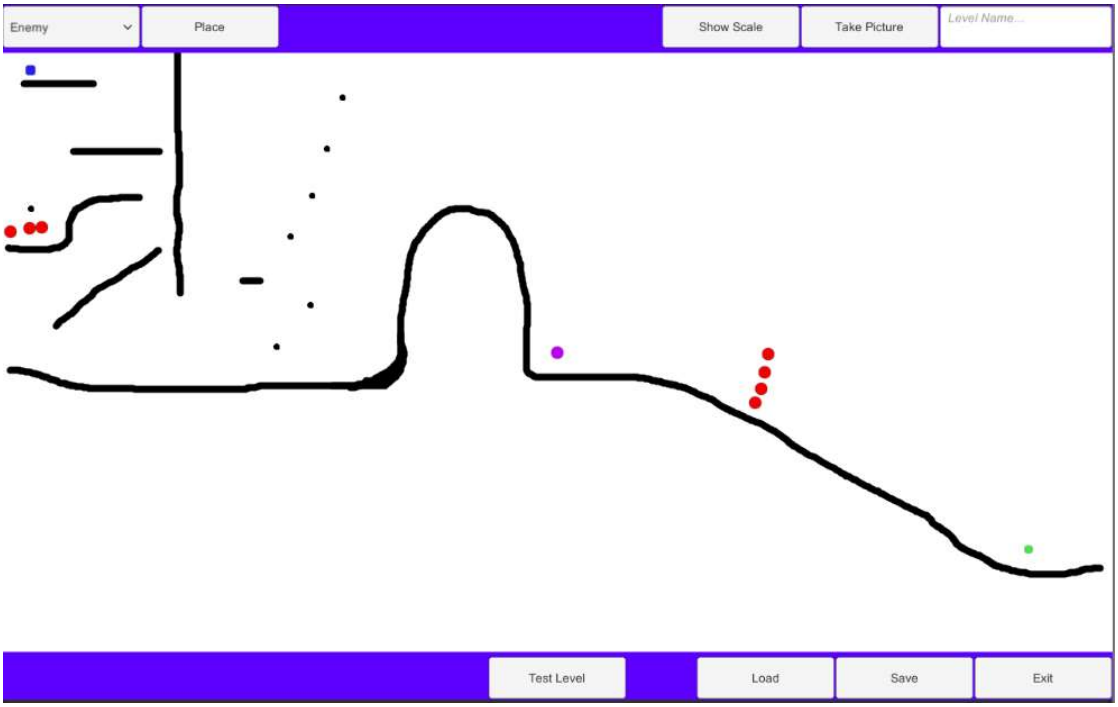


Kuvio 9. Platformeja ja niiden collidereita. Huomaa viimeisen platformin colliderista puuttuva osa.

6 Ominaisuuksien muokkaus

Platformien lisäksi kentälle voidaan ja täytyy lisätä muitakin elementtejä. Kentät ovat vapaa muotoisia, ne voivat pohjautua kiipeämiseen, sivusuunnassa etenemiseen, laskeutumiseen tai yhdistelmään edellisiä. Tästä johtuen jokaiselle kentälle täytyy antaa alku- ja loppupiste eli piste, josta pelaaja aloittaa etenemisen ja maali, johon pelaaja pyrkii pääsemään. Nämä pisteet lisätään kentälle valitsemalla valikosta lisättävä objekti ja klikkaamalla Place-nappulaa, jolloin kenttä avautuu näkymään kokonaan ja haluttu paikka valitaan klikkaamalla (ks. kuvio 10). Kentällä voi olla ainoastaan yksi

aloituspiste ja maali. Mikäli kentälle lisätään toinen aloituspiste tai maali, alkuperäinen poistetaan.



Kuvio 10. Kenttäeditorin käyttöliittymä.

Viholliset ovat objekteja, joita koskettaessa pelaaja joutuu aloittamaan kentän uudestaan aloituspisteestä. Viholliset lisätään kentälle käyttäen samaa tekniikkaa, kuin aloituspiste ja maali. Vihollisia voi olla kentällä rajoittamaton määrä, joten niille on myös poisto valinta, joka poistaa klikattua pistettä lähimmän vihollisen.

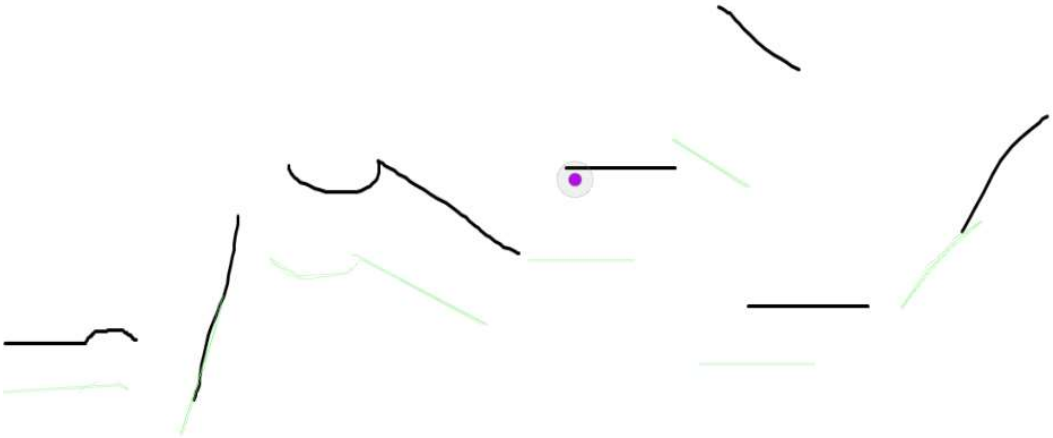
Yksi kentän ominaisuus on skaalaus eli paljonko kentästä näkyy pelaajalle yhtä aikaisesti. Skaalan asettamiseksi valitaan Show Scale, jolloin ruudulle ilmestyy suorakulmio, joka rajaa yhdelle näytölle mahtuvan alueen. Skaalaa säädetään Plus- ja Miinusnäppäimillä.

Kentän tallennus onnistuu antamalla tasolle nimi ja painamalla Save. Kentän data tallennetaan binäärisessä muodossa. Kentän dataan kuuluvat aloituspiste, maali, viholliset sekä polku kentän kuvaan. Tehdyn kentän lataaminen onnistuu antamalla kentän nimi ja painamalla Load.

Kentän pelaamiseksi painetaan Test Level, jolloin pelaaja pääsee pelaamaan luomaansa kenttää. Pelaamisen lopettamiseksi painetaan Back-nappulaa.

7 Ongelmia

Kehitysprosessissa suurimmat vastaan tulleet ongelmat liittyivät kentän luomiseen. Teoria oli vahvalla pohjalla, mutta ongelmaksi muodostui ensimmäisestä versiosta alkaen kuvasta luettujen platformien siirtäminen kuvan koordinaatistosta ja skaalasta pelimaailman erilaiseen koordinaatistoon ja skaalaan (Kuvio 3). Lopullisessa versiossakin on ongelmia skaalaamisen kanssa. Ainoastaan 1280x800 kuvasuhteessa olevat kuvat skaalautuvat oikein (ks. kuvio 11).



Kuvio 11. 1200x600 kokoisesta kuvasta generoidut platformit ja niiden colliderit.

Ehdottomasti suurin ongelma oli kehityksen pätkiminen ja aikataulutuksen epäonnistuminen. Alkuperäisen suunnitelman mukaan työn piti olla valmis dokumentaatioiden kera syyskuussa 2015. Suunnitelmassa yliarvioitiin kehityksen nopeus sekä kehittäjän päivätyön viemä aika ja energia aliarvioitiin. Kesän lopussa töiden pysähdyttyä kokonaan projekti lähes jätettiin saattamatta päätökseen. Projektia päätettiin jatkaa vasta täyden vuoden kuluttua 2016 marraskuussa, jolloin tehtiin toteutus Unityä käyttäen. Viimeiseen tilaansa projekti saatettiin 2017 toukokuussa, jolloin myös kirjoitettiin valtaosa dokumentaatiosta.

8 Tulokset, pohdinta ja jatkokehitys

Työn tavoitteena oli kehittää peli, joka voitaisiin julkaista yleiseen käyttöön. Tätä tavoitetta ei kuitenkaan saavutettu. Lopputuloksena on prototyyppejä, joilla voidaan

luoda kuvista kenttiä. Vaikka työssä käytettiin nyt jo vanhentuneita keinoja, pystyttiin suurin haaste toteuttamaan ja siten osoittamaan, että on mahdollista luoda kenttiä käyttäen yksinkertaisia kuvia lähteenä, mikä oli tärkein osa työtä. Kehitysprosessin jakautuminen useaan aliprojektiin ja niiden kehityksen ajoituksien erot hidastivat projektin etenemistä kokonaisuutena.

Projektin ydintulos toimii hyvänä pohjana mahdollisille tuleville projekteille, jotka hyödyntävät kuvasta lukua. Logiikan siirtäminen uuteen pohjaan käyttäen moderneja työkaluja mahdollistaa uuden tyyppisen kenttien luomisen, mitä ei ole vielä nähty valmiissa tuotteissa. Kuvan luomisen käyttäjäystävällisyyttä voidaan todennäköisesti parantaa. Mahdollisesti käyttämällä hienostunutta reunantunnistusta yhdistettynä käyttäjälähtöiseen virheiden poistoon. Tämä sallisi maisemakuvien käytön lähdekuvana, mutta tekisi editorin käytöstä monivaiheisempaa ja vaikeampaa, mikä on ristiriidassa tämän projektin tavoitteiden kanssa.

Pelinkehitys tuotteeksi epäonnistui, mikä on jatkokehityksen kannalta huono asia. Mikäli DIY-Platformeria kehitettäisiin eteenpäin, projekti olisi parasta aloittaa uudelta pohjalta uudistaen koko koodipohjan, käyttäen uutta kuvankäsittelykirjastoa. Projektia voidaan kuitenkin jatkaa eri tavoitetta kohti. Työstä voidaan tehdä pienellä jatkokehityksellä versio, jolla voidaan osoittaa alkuperäisen kaltaisen tuotteen olevan mahdollinen. Tällaista prototyyppiä voidaan esitellä alan yrityksille, joilta saatava tuki mahdollistaisi uuden, tuotteeksi asti kehitettävän projektin aloittamisen.

Lähteet

International Telecommunication Union. 2015. Parameter values for the HDTV standards for production and international programme exchange. Suositus. 4. Viitattu 30.5.2017, https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf

Kirillov, A. 2013. AForge.NET-framework. Verkkosivu. Viitattu 30.5.2017 http://www.aforgenet.com/framework/

South, J. ImageProcessor. Verkkosivu. Viitattu 30.5.2017 http://imageprocessor.org/